

Programování (v C)

Jan Faigl

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Přednáška 02

B0B36PRP – Procedurální programování

Jan Faigl, 2025

B0B36PRP – Přednáška 02: Programování (v C)

1 / 59

Program v C

Funkce

Číselné typy

Literály

Výrazy a operátory

Část I

Část 1 – Programování v C

Jan Faigl, 2025

B0B36PRP – Přednáška 02: Programování (v C)

3 / 59

Přehled témat

■ Část 1 – Programování v C

- Program v C
- Funkce
- Číselné typy
- Literály
- Výrazy a operátory

S. G. Kochan: kapitoly 2, 3, 4

■ Část 2 – Řídící struktury (úvod)

- Řídící struktury
- Složený příkaz
- Větvení
- Cykly

S. G. Kochan: kapitola 5 a část kapitoly 6

■ Část 3 – Zadání 2. domácího úkolu (HW02)

Jan Faigl, 2025

B0B36PRP – Přednáška 02: Programování (v C)

2 / 59

Program v C

Funkce

Číselné typy

Literály

Výrazy a operátory

Zdrojové soubory programu v C

■ Zdrojový soubor s koncovkou .c.

Zpravidla—základní rozlišení souborů, viz např. .C.

■ Hlavičkový soubor s koncovkou .h.

Jména souborů volíme výstižné (krátké názvy) a zpravidla zapisujeme malými písmeny.

■ Zdrojové soubory jsou překládány do binární podoby překladačem a vznikají objektové soubory (.o) nebo spustitelný program.

Objektový kód obsahuje relativní adresy proměnných a volání funkcí nebo pouze odkazy na jména funkcí, jejichž implementace ještě nemusejí být známy.

■ Z objektových souborů (object files) se sestavuje výsledný program, ve kterém jsou již všechny funkce známy a relativní adresy se nahradí absolutními.

Program se zpravidla sestavuje z více objektových souborů umístěných například v knihovnách.

Jan Faigl, 2025

B0B36PRP – Přednáška 02: Programování (v C)

5 / 59

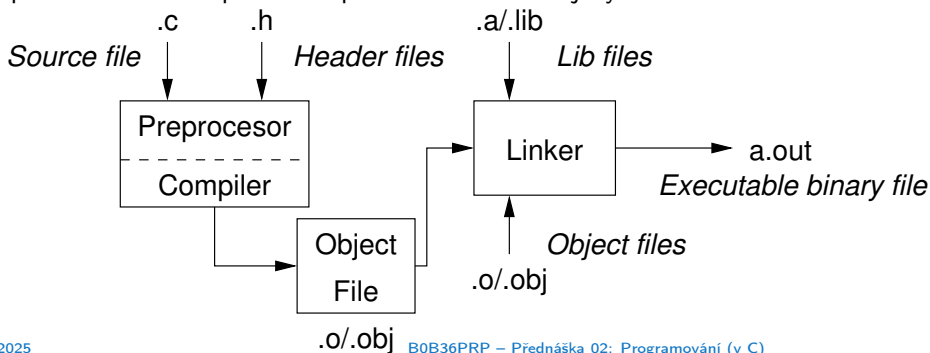
Jan Faigl, 2025

B0B36PRP – Přednáška 02: Programování (v C)

5 / 59

Schéma překladač a sestavení programu

- Vývoj programu se skládá z editace zdrojových souborů (.c a .h).
 - Kompilace zdrojových souborů (.c) do objektových (.o nebo .obj).
 - **Preprocesor** – zpracování maker a přizpůsobení překladač kompilačnímu prostředí.
 - Linkování přeložených (objektových) souborů do spustitelného programu.
 - Také vytváření dynamicky linkovaných knihoven.
- Spouštění a ladění aplikace a opětovné editace zdrojových souborů.



Zdrojové soubory

Proč .c a .h soubory?

- Zdrojové a hlavičkové soubory umožňují rozlišit **deklaraci** a **definici** podporující:
 - **Znovupoužitelnost**
 - K využití binární knihovny potřebuje znát „rozhraní“ funkcí (případně typů), které je deklarované v hlavičkovém souboru.
 - Např. funkce standardní knihovny C, libc.
 - **Modularitu**
 - Hlavičkový soubor obsahuje popis co modul nabízí, tj. popis (seznam) funkcí a jejich parametrů (**deklarace funkcí**) bez konkrétní implementace.
 - Implementace funkce je **definice funkce**.
 - **Organizaci** zdrojových kódů v adresářové struktuře souborů.

Deklarování, že funkce existují a jaké mají rozhraní (vstup a výstup) argumenty a návratový typ udávající velikost paměti pro předávaná data.

Pro jednoduché programy a domácí úkoly nedává moc smysl.

Vyplatí se především v HW08 a HW10, případně HW06 (Matice)!

Překladače jazyka C

- V PRP používáme **gcc** a **clang** (C language family frontend for LLVM).
 - <https://gcc.gnu.org> <http://clang.llvm.org>
 - Základní použití (přepínače a argumenty) je u obou překladačů stejné.
- Příklad použití
 - **compile:** `gcc -c program.c -o program.o`
 - **link:** `gcc program.o -o program`
 - Sloučení překladač a sestavení v jediném příkazu `clang program.c -o program`
- Informace o souboru (**file**) a závislosti na dynamických knihovnách (**ldd**).

```
% clang var.c -o var
% ldd var
var:
    libc.so.7 => /lib/libc.so.7 (0x2c41d000)
```

```
% file var
var: ELF 64-bit LSB executable, x86-64, version 1 (
FreeBSD), dynamically linked, interpreter /
libexec/ld-elf.so.1, for FreeBSD 12.4
(1204500), FreeBSD-style, with debug_info, not
stripped
```

Spustitelný program – main()

- Spustitelný program musí obsahovat jedinou definici funkce **main()**, která má základní tvary předání argumentů programu.

```
int main(int argc, char *argv[])    int main(int argc, char **argv, char **envp)
{
    ...
}
```

- Při spuštění programu předává OS programu počet argumentů (**argc**) a argumenty (**argv**), jako pole textových řetězců.
 - První argument je jméno programu.
- Návratová hodnota je předána OS, kde je možné ji dále použít.

```
1 int main(int argc, char *argv[])
2 {
3     int v;
4     v = 10;
5     v = v + 1;
6     return argc;
7 }
```

```
■ Návratová hodnota programu je v proměnné $? . sh, bash, zsh
■ Příklad spuštění s různým počtem argumentů.
./var
./var; echo $?
1
./var 1 2 3; echo $?
4
./var a; echo $?
2
```

lec02/var.c

Příklad kompilace a spuštění programu

- Sestavení programu `clang var.c` automaticky dojde ke kompilaci a linkování programu do spustitelného souboru `a.out`.
Výchozí jméno programu.
- Výstupní (output) soubor specifikujeme `clang var.c -o var` a spustíme, např. `./var`.
- Kompilaci a spuštění můžeme spojit do dvojice příkazů `clang var.c -o var; ./var`.
- Spuštění můžeme podmínit úspěšnou kompilací programu `clang var.c -o var && ./var`.
Návratová hodnota programu — 0 (EXIT_SUCCESS) znamená OK, chyb může být více.
Logický operátor && závisí na příkazovém interpretu, např. sh, bash, zsh.
- Příznakem `-E` můžeme při „kompilaci“ vyvolat pouze preprocesor: `gcc -E var.c`.

```
1 # 1 "var.c"
2 # 1 "<built-in>"
3 # 1 "<command-line>"
4 # 1 "var.c"
5 int main(int argc, char **argv) {
6     int v;
7     v = 10;
8     v = v + 1;
9     return argc;
10 }
```

Vlastnosti funkcí

- C nepovoluje funkce vnořené do jiných funkcí.
- Jména funkcí se mohou exportovat do ostatních modulů.
Modul je samostatně překládaný soubor.
- Funkce jsou implicitně deklarovány jako `extern`, tj. viditelné.
- Specifikátorem `static` před jménem funkce omezíme viditelnost jména funkce pouze pro daný modul (tj. konkrétní jméno souboru `.c`).
Lokální funkce modulu.
- Formální parametry funkce jsou **lokální proměnné**, které jsou inicializovány skutečnými parametry při volání funkce.
Parametry se do funkce předávají hodnotou (Call by Value).
- **C dovoluje rekurzi** – lokální proměnné jsou pro každé jednotlivé volání zakládány znovu na zásobník.
Kód funkce v C je reentrantní ve smyslu volání funkce ze sebe sama.
- Funkce nemusí mít žádné vstupní parametry, zapisujeme klíčovým slovem `void`.
`fce(void)`
- Funkce nemusí vracet funkční hodnotu–návratový typ je `void`.
`void fce(void)`

Funkce

- Funkce tvoří základní stavební blok **modulárního** jazyka C.
Modulární program je složen z více modulů/zdrojových souborů.
- Každý spustitelný program v C obsahuje *alespoň* jednu funkci a to funkci `main()`.
 - Běh programu začíná funkcí `main()`.
- **Deklarace** se skládá z hlavičky funkce.
`typ_návratové_hodnoty jméno_funkce(seznam parametrů);`
*C používá **prototyp (hlavičku) funkce** k **deklaraci** informací nutných pro překlad tak, aby mohlo být přeloženo správné volání funkce i v případě, že **definice** je umístěna dále v kódu.*
- **Definice** funkce obsahuje **hlavičku funkce a její tělo**, syntax:
`typ_návratové_hodnoty jméno_funkce(seznam parametrů)`
`{`
 `//tělo funkce`
`}`

Definice funkce bez předchozí deklarace je zároveň deklarací funkce.

Příkaz return

- Příkaz ukončení funkce `return vyraz;`.
- `return` lze použít pouze v těle funkce.
- `return` ukončí funkci, vrátí návratovou hodnotu funkce určenou hodnotou `vyraz` a předá řízení volající funkci.
- `return` lze použít v těle funkce vícekrát.
Kódovací konvence může doporučovat nejvýše jeden výskyt return ve funkci.
- U funkce s prázdným návratovým typem, např. `void fce()`, nahrazuje uzavírací závorka těla funkce příkaz `return;`.

```
void fce(int a)
{
    ...
}
```

Argumenty funkce

- Argumenty funkce se předávají hodnotou.

```
1 int main(void)
2 {
3     int v1 = 10;
4     int v2 = 20;
5     printf("v1: %i v2: %i\n", v1, v2);
6     swap0(v1, v2);
7     printf("v1: %i v2: %i\n", v1, v2);
8     swap(&v1, &v2); //předání paměťového místa
9     printf("v1: %i v2: %i\n", v1, v2);
10    return 0;
11 }
12 }
```

```
14 void swap0(int a, int b)
15 {
16     int t = a; // dočasná proměnná
17     a = b;
18     b = t;
19 }
21 void swap(int *a, int *b)
22 {
23     int t = *a; // dočasná proměnná
24     *a = *b;
25     *b = t;
26 }
```

lec02/swap.c

- Proto předáváme adresu paměťového místa (pointer/ukazatel) – &v1 a &v2.

```
% clang swap.c -o swap && ./swap
v1: 10 v2: 20
v1: 10 v2: 20
v1: 20 v2: 10
```

min_max() – příklad volání

- Vytvoříme vstupní soubor s pěti náhodnými čísly od 1 do 99: shuf -i 1-99 -n 5.
- Standardní výstup programu shuf přesměrujeme do souboru in.txt.
- Standardní vstup našeho programu minmax přesměrujeme ze souboru in.txt.
- Vytiskneme návratovou hodnotu volání programu.

```
% clang min_max.c -o minmax
% shuf -i 1-99 -n 5 > in.txt
% ./minmax <in.txt
Read 5 numbers, min: 1, max: 9
% echo $?
0
```

lec02/min_max.c

- Vytvoříme alternativní (chybný) vstup, nebo zadáme ručně.

```
% echo "a" >in2.txt
% lec02 cat in.txt >>in2.txt
% ./minmax <in2.txt
ERROR: No input given!
```

Argumenty funkce a návratová hodnota

- K „vracení“ více hodnot, můžeme využít předání paměťových míst. *Podobně jako scanf().*
- Příklad načtení celých čísel typu int a určení minimální a maximální hodnoty.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <limits.h>
4 void min_max(int v, int *min, int *max);
5 int main(void)
6 {
7     int ret = EXIT_SUCCESS;
8     int min = INT_MAX; // limits
9     int max = INT_MIN; // limits
10    int c = 0;
11    int v;
12    while (scanf("%i", &v) == 1) {
13        min_max(v, &min, &max);
14        c = c + 1;
15    }
16 }
```

```
19 if (c > 0) {
20     printf("Read %d numbers, min: %d, max: %d\n",
21           c, min, max);
22 } else {
23     fprintf(stderr, "ERROR: No input given!\n");
24     ret = EXIT_FAILURE;
25 }
26 return ret;
27 }
28 void min_max(int v, int *min, int *max)
29 {
30     if (v < *min) *min = v;
31     if (v > *max) *max = v;
32 }
```

lec02/min_max.c

Číselné typy

- Celočíselné typy – int, long, short, char.
char – celé číslo v rozsahu jednoho bajtu nebo také znak.
- Velikost paměti alokované příslušnou (celo)číselnou proměnnou se může lišit dle architektury počítače nebo překladače.
Typ int má zpravidla velikost 4 bajty a to i na 64-bitových systémech.
- Aktuální velikost paměťové reprezentace lze zjistit operátorem sizeof(), kde argumentem je jméno typu nebo proměnné.

```
1 int i;
2 printf("%lu\n", sizeof(int));
3 printf("ui size: %lu\n", sizeof(i));
```

lec02/types.c

- Neceločíselné typy – float, double
Konkrétní reprezentace je dána implementací, většinou dle standardu IEEE-754-1985.
- float – 32-bit IEEE 754
- double – 64-bit IEEE 754

http://www.tutorialspoint.com/cprogramming/c_data_types.htm

Znaménkové a neznaménkové celočíselné typy

- Celočíselné typy kromě počtu bajtů rozlišujeme na
 - **signed** – **znaménkový** (základní);
 - **unsigned** – **neznaménkový**. *Proměnná neznaménkového typu nemůže zobrazit záporné číslo.*
 - Příklad (1 byte):
 - `unsigned char`: 0 až 255;
 - `signed char`: -128 až 127.

```
1 unsigned char uc = 127;
2 char su = 127;
3
4 printf("The value of uc=%i and su=%i\n", uc, su);
5 uc = uc + 2;
6 su = su + 2;
7 printf("The value of uc=%i and su=%i\n", uc, su);
```

lec02/signed_unsigned_char.c

```
$ clang signed_unsigned.c && ./a.out
The value of uc=127 and su=127
The value of uc=129 and su=-127
```

Logický datový typ (Boolean) – `_Bool`

- Ve verzi **C99** je zaveden logický datový typ `_Bool`.
- 1 `_Bool` logic_variable;
- Jako hodnota `true` je libovolná hodnota typu `int` různá od 0.
- Dále můžeme využít hlavičkového souboru `<stdbool.h>`, kde je definován typ `bool` a hodnoty `true` a `false`.

```
1 #define false 0
2 #define true 1
3
4 #define bool _Bool
```

- V původním (ANSI) C explicitní datový typ pro logickou hodnotu není definován.
 - Můžeme však použít podobnou definici jako v `<stdbool.h>`.

```
1 #define FALSE 0
2 #define TRUE 1
```

Znak – `char`

- Znak je typ `char`.
- Znak reprezentuje celé číslo (byte).
 - Kódování znaků (grafických symbolů), např. ASCII – American Standard Code for Information Interchange.*

- Hodnotu znaku lze zapsat jako tzv. znakovou konstantu, např. `'a'`.

```
1 char c = 'a';
2
3 printf("The value is %i or as char '%c'\n", c, c);
```

lec02/char.c

```
clang char.c && ./a.out
The value is 97 or as char 'a'
```

- Pro řízení výstupních zařízení jsou definovány řídicí znaky.

Tzv. *escape sequences*

- `\t` – tabulátor (tabular), `\n` – nový řádek (newline),
- `\a` – pípnutí (beep), `\b` – backspace, `\r` – carriage return,
- `\f` – form feed, `\v` – vertical space

Rozsahy celočíselných typů

- Rozsahy celočíselných typů v C nejsou dány normou, ale implementací.
 - Mohou se lišit implementací a prostředím 16 bitů vs 64 bitů.*
- Norma garantuje, že pro rozsahy typů platí.
 - `short ≤ int ≤ long`
 - `unsigned short ≤ unsigned ≤ unsigned long`
- Pokud chceme zajistit definovanou velikost můžeme použít definované typy například z hlavičkového souboru `<stdint.h>`.

IEEE Std 1003.1-2001

<code>int8_t</code>	<code>uint8_t</code>
<code>int16_t</code>	<code>uint16_t</code>
<code>int32_t</code>	<code>uint32_t</code>

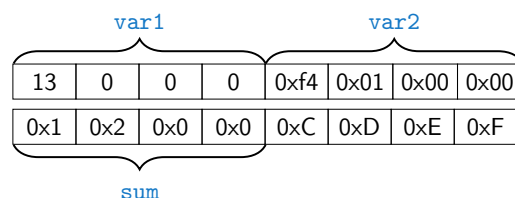
lec02/inttypes.c

<http://pubs.opengroup.org/onlinepubs/009695399/basedefs/stdint.h.html>

Přiřazení, proměnné a paměť – Vizualizace int

```
1  int var1;
2  int var2;
3  int sum;
5  // 00 00 00 13
6  var1 = 13;
8  // x00 x00 x01 xF4
9  var2 = 500;
11 sum = var1 + var2;
```

- Proměnné typu `int` alokují 4 bajty.
Zjistit velikost můžeme operátorem `sizeof(int)`.
- Obsah paměti není po alokaci definován.



500 (dec) je 0x01F4 (hex)

513 (dec) je 0x0201 (hex)

*V případě architektury Intel x86 a x86-64 jsou hodnoty uloženy v pořadí **little-endian**.*

Literály racionálních čísel

- Formát zápisu racionálních literálů:
 - S řádovou tečkou – 13.1;
 - Mantisa a exponent – 31.4e-3 nebo 31.4E-3.
- Typ racionálního literálu:
 - `double` – pokud není explicitně určen;
 - `float` – přípona `F` nebo `f`;
 - `long double` – přípona `L` nebo `l`.

```
float f = 10f;
```

```
long double ld = 101;
```

Literály

- Jazyk C má 6 typů literálů (konstantních hodnot).
 - Celočíselné;
 - Racionální;
 - Znakové;
 - Řetězcové;
 - Výčtové – *pojmenovaná celá čísla typu int*;
 - Symbolické – `#define NUMBER 10`.

Enum

Preprocesor

Znakové literály

- Formát – jeden (případně více) znaků v jednoduchých apostrofech 'A', 'B' nebo '\n'.
- Hodnota – jednoznakový literál má hodnotu odpovídající kódu znaku '0' ~ 48, 'A' ~ 65.
Hodnota znaků mimo ASCII (větší než 127) závisí na překladači.
- Typ znakové konstanty.
 - Znaková konstanta je typu `int`.
Automatická konverze kódu ASCII znaku na typ `char`.

Řetězcové literály

- Formát – posloupnost znaků a řídicích znaků (escape sequences) uzavřená v uvozovkách.
"Řetězcová konstanta s koncem řádku\n"
 - Řetězcové konstanty oddělené oddělovači (white spaces) se sloučí do jediné, např.
"Řetězcová konstanta" " s koncem řádku\n"
se sloučí do
"Řetězcová konstanta s koncem řádku\n".
- Typ
 - Řetězcová konstanta je uložena v poli typu `char` a zakončená znakem `'\0'`.
Např. řetězcová konstanta `"word"` je uložena jako posloupnost znaků/bajtů (pole).

```
'w' 'o' 'r' 'd' '\0'
```

Pole tak musí být vždy o 1 položku delší!
Více o textových řetězcích na 4. přednášce a cvičení.

Symbolické konstanty – #define

- Formát – konstanta je založena příkazem preprocesoru `#define`.
 - Je to makro příkaz bez parametru.
 - Každý `#define` musí být na samostatném řádku.
#define SCORE 1

Zpravidla píšeme velkými písmeny.
- Symbolické konstanty mohou vyjadřovat konstantní výraz.
#define MAX_1 ((10*6) - 3)
- Symbolické konstanty mohou být vnořené.
#define MAX_2 (MAX_1 + 1)
- **Preprocesor provede textovou náhradu definované konstanty za její hodnotu.**
#define MAX_2 (MAX_1 + 1)

Je-li hodnota výraz, jsou kulaté závorky nutné pro správné vyhodnocení výrazu, např. pro $5 * MAX_1$ s vnějšími závorkami je $5 * ((10 * 6) - 3) = 285$ vs $5 * (10 * 6) - 3 = 297$.

Konstanty výčtového typu

- Implicitní hodnoty konstanty výčtového typu začínají od 0 a hodnoty můžeme explicitně předepsat. Bez uvedení hodnoty má další prvek hodnotu o jedničku vyšší než ten předchozí.
- Výčtové konstanty mohou mít stejné hodnoty.

```
1 enum { 1 enum { 1 enum {
2 SPADES, // začíná 0 2 SPADES = 10, 2 NUMBER_1 = 1,
3 CLUBS, // hodnota je 11 3 CLUBS = 2, 3 NUMBER_2 = 2,
4 HEARTS, 4 HEARTS = 15, 4 NUMBER_3 = 1,
5 DIAMONDS 5 DIAMONDS = 13 5 NUMBER_4, // hodnota je 2
6 }; 6 }; 6 };
```

Hodnoty výčtu zpravidla píšeme velkými písmeny.

- Typ – výčtová konstanta je typu `int`.
 - Hodnotu konstanty můžeme použít pro iteraci v cyklu.
- ```
1 enum { SPADES = 0, CLUBS, HEARTS, DIAMONDS, NUM_COLORS };
3 for (int i = SPADES; i < NUM_COLORS; ++i) {
4 ...
5 }
```

lec02/enum.c

# Proměnné s konstantní hodnotou – modifikátor (const)

- Uvedením klíčového slova `const` můžeme označit proměnnou jako konstantní.  
Překladač kontroluje přiřazení a nedovolí hodnotu proměnné nastavit znovu.
- Pro definici konstant můžeme použít konstantní proměnné, symbolické konstanty (preprocesor) a v případě celočíselných hodnot (`int`) také `enum`.
- Proměnné s konstantní hodnotou mají typ a paměť  
const float pi = 3.14159265;
- na rozdíl od symbolické konstanty  
#define PI 3.14159265
- reprezentující literál.

| Program v C                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Funkce                                      | Číselné typy | Literály | Výrazy a operátory |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|--------------|----------|--------------------|
| <h2>Výrazy</h2> <ul style="list-style-type: none"> <li>■ <b>Výraz</b> předepisuje výpočet hodnoty určitého vstupu.</li> <li>■ Struktura výrazu obsahuje <b>operandy</b>, <b>operátory</b> a <b>závorky</b>.</li> <li>■ Výraz může obsahovat <ul style="list-style-type: none"> <li>■ literály,</li> <li>■ unární a binární operátory,</li> <li>■ proměnné,</li> <li>■ volání funkcí,</li> <li>■ konstanty,</li> <li>■ závorky.</li> </ul> </li> <li>■ Pořadí operací předepsaných výrazem je dáno <b>prioritou</b> a <b>asociativitou</b> operátorů.</li> </ul> <p><b>Příklad</b></p> <pre>10 + x * y    // pořadí vyhodnocení 10 + (x * y) 10 + x + y    // pořadí vyhodnocení (10 + x) + y</pre> <p style="text-align: right;">* má vyšší prioritu než +<br/>+ je asociativní zleva</p> |                                             |              |          |                    |
| Jan Faigl, 2025                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | B0B36PRP – Přednáška 02: Programování (v C) |              |          | 34 / 59            |

| Program v C                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Funkce                                      | Číselné typy | Literály | Výrazy a operátory |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|--------------|----------|--------------------|
| <h2>Proměnné, operátor přiřazení a příkaz přiřazení</h2> <ul style="list-style-type: none"> <li>■ Proměnné definujeme uvedením typu a jména proměnné. <ul style="list-style-type: none"> <li>■ Jména proměnných volíme malá písmena.</li> <li>■ Viceslovná jména zapisujeme s podtržítkem <code>_</code>.</li> <li>■ Proměnné definujeme na samostatném řádku.</li> </ul> </li> </ul> <p style="text-align: right;">Nebo volíme <i>CamelCase</i>.</p> <pre>int n; int number_of_items;</pre> <ul style="list-style-type: none"> <li>■ <b>Proměnné reprezentují data, proto volíme podstatná jména.</b></li> <li>■ Přiřazení je nastavení hodnoty proměnné, tj. uložení definované hodnoty na místo v paměti, kterou proměnná reprezentuje.</li> <li>■ Tvar <b>přiřazovacího operátoru</b>. <math display="block">\langle \text{proměnná} \rangle = \langle \text{výraz} \rangle</math> <p style="text-align: right;">Výraz je literál, proměnná, volání funkce, ...</p> </li> <li>■ <b>Příkaz přiřazení</b> se skládá z operátoru přiřazení <code>=</code> a <code>;</code>. <ul style="list-style-type: none"> <li>■ Levá strana přiřazení musí být <b>l-value – location-value, left-value</b>. <p style="text-align: right;">Tj. musí reprezentovat paměťové místo pro uložení výsledku.</p> </li> <li>■ Přiřazení je výraz a můžeme jej použít všude, kde je dovolen výraz příslušného typu.</li> </ul> </li> </ul> |                                             |              |          |                    |
| Jan Faigl, 2025                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | B0B36PRP – Přednáška 02: Programování (v C) |              |          | 36 / 59            |

| Program v C                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Funkce                                      | Číselné typy | Literály | Výrazy a operátory |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|--------------|----------|--------------------|
| <h2>Základní rozdělení operátorů</h2> <ul style="list-style-type: none"> <li>■ Operátory jsou vyhrazené znaky (nebo posloupnost znaků) pro zápis výrazů.</li> <li>■ Můžeme rozlišit čtyři základní typy <b>binárních operátorů</b>. <ul style="list-style-type: none"> <li>■ <b>Aritmetické</b> operátory – sčítání, odčítání, násobení, dělení;</li> <li>■ <b>Relační</b> operátory – porovnání hodnot (menší, větší, ...);</li> <li>■ <b>Logické</b> operátory – logický součet a součin;</li> <li>■ <b>Operátor přiřazení</b> – na levé straně operátoru <code>=</code> je proměnná.</li> </ul> </li> <li>■ <b>Unární operátory</b> <ul style="list-style-type: none"> <li>■ indukující kladnou/zápornou hodnotu: <code>+</code> a <code>-</code>;</li> </ul> <p style="text-align: right;"><i>Unární operátor minus – modifikuje znaménko výrazu za ním.</i></p> <ul style="list-style-type: none"> <li>■ modifikující proměnou <code>++</code> a <code>--</code>;</li> <li>■ logický operátor doplněk <code>!</code>;</li> <li>■ operátor přetypování (<code>jméno typu</code>).</li> </ul> </li> <li>■ <b>Ternární operátor</b> – podmíněný výsledek výrazu ze dvou výrazů. <math display="block">\text{výraz} \ ? \ \text{hodnota}_1 \ : \ \text{hodnota}_2</math> <p style="text-align: right;"><i>Hodnota výrazu ternárního operátoru je buď druhý nebo třetí operand v závislosti na logické hodnotě prvního operandu.</i></p> </li> </ul> |                                             |              |          |                    |
| Jan Faigl, 2025                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | B0B36PRP – Přednáška 02: Programování (v C) |              |          | 35 / 59            |

| Program v C                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Funkce                                      | Číselné typy | Literály | Výrazy a operátory |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|--------------|----------|--------------------|
| <h2>Základní aritmetické výrazy</h2> <ul style="list-style-type: none"> <li>■ Pro operandy (ne)celočíslných typů <code>int</code>, <code>char</code>, <code>short</code> a <code>double</code> a <code>float</code> jsou definovány operátory: <ul style="list-style-type: none"> <li>■ unární operátor změna znaménka <code>-</code>;</li> <li>■ binární sčítání <code>+</code> a odčítání <code>-</code>;</li> <li>■ binární násobení <code>*</code> a dělení <code>/</code>.</li> </ul> </li> <li>■ Pro operandy celočíselných typů pak dále <ul style="list-style-type: none"> <li>■ binární zbytek po dělení <code>%</code>.</li> </ul> </li> <li>■ Pro oba operandy stejného typu je výsledek aritmetické operace stejného typu.</li> <li>■ V případě kombinace typů <code>int</code> a <code>double</code>, se <code>int</code> převede na <code>double</code> a výsledek je hodnota typu <code>double</code>. <p style="text-align: right;"><i>Implicitní typová konverze.</i></p> </li> <li>■ Dělení operandů typu <code>int</code> je celá část podílu. <p style="text-align: right;"><i>Např. 7/3 je 2 a -7/3 je -2</i></p> </li> <li>■ Pro zbytek po dělení platí <math>x \% y = x - (x / y) * y</math>. <p style="text-align: right;"><i>Např. 7 % 3 je 1      -7 % 3 je -1      7 % -3 je 1      -7 % -3 je -1</i></p> <p style="text-align: right;"><i>Pro záporné operandy je v C99 výsledek celočíselného dělení blíže 0, platí (a/b)*b + a%b = a. Pro starší verze C závisí výsledek na překladači.</i></p> <p style="text-align: right;"><i>Další operátory přistě.</i></p> </li> </ul> |                                             |              |          |                    |
| Jan Faigl, 2025                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | B0B36PRP – Přednáška 02: Programování (v C) |              |          | 37 / 59            |

| Program v C                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Funkce                                      | Číselné typy | Literály | Výrazy a operátory |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|--------------|----------|--------------------|
| Příklad – Aritmetické operátory 1/2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                                             |              |          |                    |
| <pre>1  int a = 10; 2  int b = 3; 3  int c = 4; 4  int d = 5; 5  int result;  7  result = a - b; // rozdíl 8  printf("a - b = %i\n", result);  10 result = a * b; // násobení 11 printf("a * b = %i\n", result);  13 result = a / b; // celocíselné dělení 14 printf("a / b = %i\n", result);  16 result = a + b * c; // priorita operátoru 17 printf("a + b * c = %i\n", result);  19 printf("a * b + c * d = %i\n", a * b + c * d);          // -&gt; 50 20 printf("(a * b) + (c * d) = %i\n", (a * b) + (c * d)); // -&gt; 50 21 printf("a * (b + c) * d = %i\n", a * (b + c) * d);    // -&gt; 350                                  lec02/arithmetic_operators.c</pre> |                                             |              |          |                    |
| Jan Faigl, 2025                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | B0B36PRP – Přednáška 02: Programování (v C) |              |          | 38 / 59            |
| Řídicí struktury                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Složený příkaz                              | Větvění      | Cykly    |                    |

## Část II

### Část 2 – Řídicí struktury

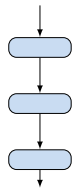
| Program v C                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Funkce                                      | Číselné typy | Literály | Výrazy a operátory |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|--------------|----------|--------------------|
| Příklad – Aritmetické operátory 2/2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                             |              |          |                    |
| <pre>1  #include &lt;stdio.h&gt; 2  int main(void) 3  { 4      int x1 = 1; 5      double y1 = 2.2357; 6      float x2 = 2.5343f; 7      double y2 = 2;  10     printf("P1 = (%i, %f)\n", x1, y1); 11     printf("P1 = (%i, %i)\n", x1, (int)y1); 12     printf("P1 = (%f, %f)\n", (double)x1, (double)y1); // operator přetypování 13     printf("P1 = (%.3f, %.3f)\n", (double)x1, (double)y1); 14     printf("P2 = (%f, %f)\n", x2, y2);  15     double dx = (x1 - x2); // implicitní konverze na float, resp. double 16     double dy = (y1 - y2);  17     printf("(P1 - P2)=(%.3f, %.3f)\n", dx, dy); 18     printf(" P1 - P2 ^2=%.2f\n", dx * dx + dy * dy); 19     return 0; 20 }</pre> |                                             |              |          |                    |
| Jan Faigl, 2025                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | B0B36PRP – Přednáška 02: Programování (v C) |              |          | 39 / 59            |
| Řídicí struktury                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Složený příkaz                              | Větvění      | Cykly    |                    |

## Řídicí struktury

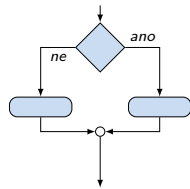
- Řídicí struktura je programová konstrukce, která se skládá z dílčích příkazů a předepisuje pro ně způsob provedení.
- Tři základní druhy řídicích struktur:
  - **Posloupnost** – předepisuje **postupné provedení** dílčích příkazů;
  - **Větvění** – předepisuje provedení dílčích příkazů v závislosti na **splnění určité podmínky**;
  - **Cyklus** – předepisuje **opakované provedení** dílčích příkazů v závislosti na splnění určité podmínky.

## Typy řídicích struktur

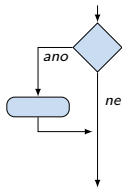
### ■ Sekvence



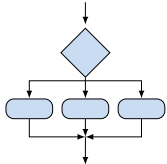
### ■ Podmínka If



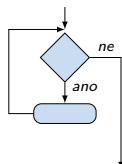
### ■ Podmínka If



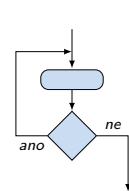
### ■ Větvení switch



### ■ Cyklus for a while



### ■ Cyklus do



## Větvení if

■ Příkaz **if** umožňuje větvení programu na základě podmínky.

■ Má dva základní tvary.

■ **if** (*podmínka*) **příkaz**<sub>1</sub>

■ **if** (*podmínka*) **příkaz**<sub>1</sub> **else** **příkaz**<sub>2</sub>

■ **podmínka** je logický výraz, jehož hodnota je logického (celočíslného) typu.

*Tj. false (hodnota 0) nebo true (hodnota různá od 0).*

■ **příkaz** je příkaz, složený příkaz nebo blok.

■ Ukázka zápisu zjištění menší hodnoty z *x* a *y*.

Varianta zápisu 1

Varianta zápisu 2

Varianta zápisu 3

```
1 int min = y;
2 if (x < y) min = x;
```

```
1 int min = y;
2 if (x < y)
3 min = x;
```

```
1 int min = y;
2 if (x < y) {
3 min = x;
4 }
```

*Příkaz je zakončen středníkem ;*

*Která varianta splňuje kódovací konvenci a proč?*

## Složený příkaz a blok

■ Řídicí struktury mají obvykle formu strukturovaných příkazů.

■ **Složený příkaz** – posloupnost příkazů.

■ **Blok** – posloupnost definic proměnných a příkazů.

```
1 {
2 //blok je vymezen složenými závorkami
3 int steps = 10;
4 printf("No. of steps %i\n", steps);
5 }
6
7 steps += 1; //nelze - mimo rozsah platnosti bloku
```

*Definice – alokace paměti podle konkrétního typu proměnné. Rozsah platnosti proměnné je lokální v rámci bloku.*

■ Budeme používat složené příkazy:

■ složený příkaz nebo blok pro posloupnost;

■ příkaz **if** nebo **switch** pro větvení;

■ příkaz **while**, **do** nebo **for** pro cyklus.

*Podmíněné opakování bloku nebo složeného příkazu.*

■ **Funkce** je **pojmenovaný blok příkazů**, který můžeme **znovupoužít**.

## Příklad větvení if

**Příklad:** Jestliže *x < y* vyměňte hodnoty těchto proměnných

Nechť proměnné *x* a *y* jsou definovány a jsou typu *int*.

Varianta 1

Varianta 2

Varianta 3

Varianta 4

```
1 if (x < y)
2 tmp = x;
3 x = y;
4 y = tmp;
```

```
1 if (x < y)
2 int tmp = x;
3 x = y;
4 y = tmp;
```

```
1 int tmp;
2 if (x < y)
3 tmp = x;
4 x = y;
5 y = tmp;
```

```
1 if (x < y) {
2 int tmp = x;
3 x = y;
4 y = tmp;
5 }
```

■ Která varianta je správně a proč?

## Příklad větvení if-then-else

**Příklad:** Do proměnné *min* uložte menší z čísel *x* a *y* a do *max* uložte větší z čísel.

Nechť proměnné *x*, *y*, *min* a *max* jsou definovány a jsou typu *int*.

### Varianta 1

```
1 if (x < y)
2 min = x;
3 max = y;
4 else
5 min = y;
6 max = x;
```

### Varianta 2

```
1 if (x < y) {
2 min = x;
3 max = y;
4 } else {
5 min = y;
6 max = x;
7 }
```

- Která varianta odpovídá našemu zadání?

## Příklad cyklu while

- Základní příkaz cyklu **while** má tvar **while (podmínka) příkaz**.

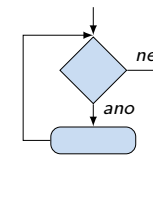
### Příklad

```
1 int x = 10;
2 int y = 3;
3 int q = x;
4 while (q >= y) {
5 q = q - y;
6 }
7 }
```

- Jaká je hodnota proměnné *q* po skončení cyklu?

## Cyklus while ()

- Příkaz **while** má tvar **while (vyraz) prikaz**;
- Příkaz cyklu **while** probíhá:
  1. Vyhodnotí se výraz **vyraz**;
  2. Pokud **vyraz != 0**, provede se příkaz **prikaz**, jinak cyklus končí;
  3. Opakování vyhodnocení výrazu **vyraz**.
- Řídicí cyklus se vyhodnocuje na začátku cyklu, cyklus se nemusí provést ani jednou.
- Řídicí výraz **vyraz** se musí aktualizovat v těle cyklu, jinak je cyklus nekonečný.

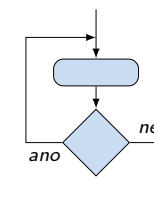


Příklad zápisu

```
int i = 0;
while (i < 5) {
 ...
 i += 1;
}
```

## Cyklus do...while ()

- Příkaz **do...while ()** má tvar **do prikaz while (vyraz)**;
- Příkaz cyklu **do...while ()** probíhá:
  1. Provede se příkaz **prikaz**;
  2. Vyhodnotí se výraz **vyraz**;
  3. Pokud **vyraz != 0**, cyklus se opakuje provedením příkazu **prikaz**, jinak cyklus končí.
- Řídicí cyklus se vyhodnocuje na konci cyklu, tělo cyklu se vždy provede nejméně jednou.
- Řídicí výraz **vyraz** se musí aktualizovat v těle cyklu, jinak je cyklus nekonečný.



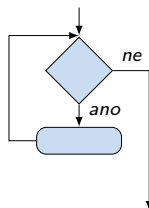
Příklad zápisu

```
1 int i = -1;
2 do {
3 ...
4 i += 1;
5 } while (i < 5);
```

## Cyklus for

- Základní příkaz cyklu **for** má tvar **for** (*inicializace; podmínka; změna*) příkaz.
- Odpovídá cyklu while v následujícím tvaru.

```
inicializace;
while (podmínka) {
 příkaz;
 změna;
}
```



### Příklad

```
1 for (int i = 0; i < 10; ++i) {
2 printf("i: %i\n", i);
3 }
```

- Změnu řídicí proměnné lze zkráceně zapsat operátorem inkrementace **++** nebo dekrementace **--**.
- Alternativně lze též použít zkrácený zápis přiřazení, např. **+=**.

## Cyklus for – příklady

- Jak se změní výstup když použijeme místo prefixového zápisu **++i** postfixový zápis **i++**.

```
1 for (int i = 0; i < 10; i++) {
2 printf("i: %i\n", i);
3 }
```

- V cyklu můžeme také řídicí proměnou dekrementovat.

```
1 for (int i = 10; i >= 0; --i) {
2 printf("i: %i\n", i);
3 }
```

Kolik program vypíše řádků?

- Kolik řádků vypíše program?

```
1 for (int i = 10; i > 0; --i) {
2 printf("i: %i\n", i);
3 }
```

- Řídicí proměnná může být také neceločíselného typu, např. **double**.

```
1 #include <math.h>
3 for (double d = 0.5; d < M_PI; d += 0.1) {
4 printf("d: %f\n", d);
5 }
```

M\_PI je symbolická konstanta definovaná v **math.h**.

## Zadání 2. domácího úkolu HW02

### Téma: První cyklus

Povinné zadání: **3b**; Volitelné zadání: **není**; Bonusové zadání: **není**

- **Motivace:** „Automatizovat“ a zobecnit výpočet pro „libovolně“ dlouhý vstup.
- **Cíl:** Osvojit si využití cyklů jako základní programové konstrukce pro hromadné zpracování dat.
- **Zadání:** <https://cw.fel.cvut.cz/wiki/courses/b0b36prp/hw/hw02>
  - Zpracování **libovolně dlouhé** posloupnosti celých čísel.
  - Výpis načtených čísel.
  - Výpis statistiky vstupních čísel.
    - Počet načtených čísel; Počet kladných a záporných čísel a jejich procentuální zastoupení na vstupu.
    - Četnosti výskytu sudých a lichých čísel a jejich procentuální zastoupení na vstupu.
    - Průměrná, maximální a minimální hodnota načtených čísel.
- **Termín odevzdání:** **18.10.2025, 23:59:59 PDT**.

PDT – Pacific Daylight Time

## Část III

### Část 3 – Zadání 2. domácího úkolu (HW02)

Shrnutí přednášky

Diskutovaná témata

- Programování v C
  - Zápis programu v C
  - Program, zdrojové soubory a kompilace programu
  - Literály a konstantní hodnoty
  - Proměnné, základní číselné typy
  - Proměnné, přiřazení a paměť
  - Základní výrazy
  - Řídící struktury
- Příště: Dokončení řídících struktur, výrazy.

Část V  
Appendix

Kódovací příklad – Zadání

- Implementujte program, který vytiskne vzor o sedmi řádcích.

1

2

3

4

5

6

7

\*\*\*\*\*

\*\* \*\* \*\*

\*\*\* \*\* \*

\*\*\*\*\*

\*\*\* \*\* \*

\*\* \*\* \*

\*\*\*\*\*
- Výchozí šířka  $n$  je 27 znaků nebo je načtena jako první argument programu (je-li zadán).
- Šířka  $n$  musí být liché číslo, jinak program vrátí 100.
- Platí  $11 \leq n \leq 67$ , jinak program vrátí 101.
- Při úspěchu program vytiskne sedm řádků a vrátí 0 (EXIT\_SUCCESS).
- Snažte se maximálně vyhnout použití "magic numbers" v programu.
- Argument programu `argv[1]` převedte na číslo `atoi()`, jeli zadán.
- Dekomponujte program jako tisk 7x řádků.
- Implementujte „infrastrukturu“ programu.
- Následně řešte logiku jednotlivých řádků řízených vhodně navrženým výrazem.

Příklad kódování – Strategie implementace 1/4

- Definujeme návratové (chybové) hodnoty (0, 100, 101) využitím `enum`, aby byl „kód čistý“.
- Definujeme platný rozsah <11,67>, (`#define`).
- Zajistíme přístup k argumentům programu pouze tehdy, pokud jsou zadány.
- Kontrolujeme, že počet řádků *n* je platná hodnota, jinak program vrací chybu.
- Provádíme libovolnou operaci pouze v případě, že argumenty (hodnoty) jsou platné.
- Tisk 7-mi řádků rozdělíme do dvou `for` smyček, mezi smyčkami bude tisk plného \* řádku.
- Implementujeme samostatnou funkci tisk vzoru řádku.

```
1 #include <stdio.h> //for putchar()
2 #include <stdlib.h> //for atoi()
3
4 enum {
5 ERROR_OK = 0,
6 ERROR_INPUT = 100,
7 ERROR_RANGE = 101
8 };
9
10 #define MIN_VALUE 11
11 #define MAX_VALUE 67
12
13 #define LINES 3
14
15 // Print line of the with n using character in c
16 // and space; with k continuous characters c
17 // followed by space.
18 void print(char c, int n, int k);
```

Příklad kódování – Strategie implementace 2/4

- Definujeme návratové (chybové) hodnoty (0, 100, 101) využitím `enum`, aby byl „kód čistý“.
- Definujeme platný rozsah <11,67>, (`#define`).
- Zajistíme přístup k argumentům programu pouze tehdy, pokud jsou zadány.
- Kontrolujeme, že počet řádků *n* je platná hodnota, jinak program vrací chybu.
- Provádíme libovolnou operaci pouze v případě, že argumenty (hodnoty) jsou platné.
- Tisk 7-mi řádků rozdělíme do dvou `for` smyček, mezi smyčkami bude tisk plného \* řádku.
- Implementujeme samostatnou funkci tisk vzoru řádku.

```
1 ...
2 int main(int argc, char *argv[])
3 {
4 int ret = ERROR_OK;
5 int n = argc > 1 ? atoi(argv[1]) : 27; //
6 // convert argv[1] or use default value
7 // n is odd number
8 ret = n % 2 == 0 ? ERROR_INPUT : ret; // ensure
9 // n is odd number
10 if (!ret &&
11 (n < MIN_VALUE || n > MAX_VALUE)) {
12 ret = ERROR_RANGE; //ensure n is in the
13 // closed interval [MIN_VALUE, MAX_VALUE]
14 }
15 ...
16 return ret;
17 }
```

Příklad kódování – Strategie implementace 3/4

- Definujeme návratové (chybové) hodnoty (0, 100, 101) využitím `enum`, aby byl „kód čistý“.
- Definujeme platný rozsah <11,67>, (`#define`).
- Zajistíme přístup k argumentům programu pouze tehdy, pokud jsou zadány.
- Kontrolujeme, že počet řádků *n* je platná hodnota, jinak program vrací chybu.
- Provádíme libovolnou operaci pouze v případě, že argumenty (hodnoty) jsou platné.
- Tisk 7-mi řádků rozdělíme do dvou `for` smyček, mezi smyčkami bude tisk plného \* řádku.
- Implementujeme samostatnou funkci tisk vzoru řádku.

```
1 // print a line with n characters with the
2 // pattern: k-times c, then space.
3 // the line ends by new line character '\n'.
4 void print(char c, int n, int k);
5
6 int main(int argc, char *argv[])
7 { ...
8 if (!ret) { // only if ret == ERROR_OK
9 for (int l = 1; l <= LINES; ++l) {
10 print('*', n, l); // print l x '*'
11 }
12 print('*', n, n); // print n x '*'
13 for (int l = LINES; l > 0; --l) {
14 print('*', n, l); // print l x 'x'
15 }
16 }
17 return ret;
18 }
```

Příklad kódování – Strategie implementace 4/4

- Definujeme návratové (chybové) hodnoty (0, 100, 101) využitím `enum`, aby byl „kód čistý“.
- Definujeme platný rozsah <11,67>, (`#define`).
- Zajistíme přístup k argumentům programu pouze tehdy, pokud jsou zadány.
- Kontrolujeme, že počet řádků *n* je platná hodnota, jinak program vrací chybu.
- Provádíme libovolnou operaci pouze v případě, že argumenty (hodnoty) jsou platné.
- Tisk 7-mi řádků rozdělíme do dvou `for` smyček, mezi smyčkami bude tisk plného \* řádku.
- Implementujeme samostatnou funkci tisk vzoru řádku.

```
1 void print(char c, int n, int k)
2 {
3 for (int i = 0; i < n; ++i) {
4 putchar((i+1) % (k+1) ? c : ' ');
5 }
6 putchar('\n');
7 }
```

- Řádek se skládá z *n* znaků, takže je třeba vypsát *n* znaků.
- Za každým *k*-tým znakem *c* je mezera.
- Násobek *k* lze zjistit ze zbytku po celočíselném dělení, operátor `%`.
- Ošetříme, že *i* začíná od 0.
- Mezera je každý *(k+1)*-tý znak.

## Příklad kódování – Strategie implementace 4(b)/4

- Definujeme návratové (chybové) hodnoty (0, 100, 101) využitím `enum`, aby byl „kód čistý“.
- Definujeme platný rozsah (11, 67), (`#define`).
- Zajistíme přístup k argumentům programu pouze tehdy, pokud jsou zadány.
- Kontrolujeme, že počet řádků  $n$  je platná hodnota, jinak program vrací chybu.
- Provádíme libovolnou operaci pouze v případě, že argumenty (hodnoty) jsou platné.
- Tisk 7-mi řádků rozdělíme do dvou `for` smyček, mezi smyčkami bude tisk plného \* řádku.
- Implementujeme samostatnou funkci tisk vzoru řádku.
- Použijeme extra proměnnou `j` pro tisk mezery, jako každý `k`-tý vytištěný znak.
- Využijeme operátor čárky k inkrementaci `j` v rámci smyčky `for`.

```
1 void print(char c, int n, int k)
2 {
3 int i, j;
4 for (i = j = 0; i < n; ++i, ++j) {
5 if (j == k) {
6 putchar(' ');
7 j = 0;
8 } else {
9 putchar(c);
10 }
11 }
12 putchar('\n');
13 }
```